https://www.youtube.com/watch?v=1MuILtHClQY&list=PLw7h5qVWrsNxjI8kbPr6nCo0l1p0253K6&index=2

Workspace:

* Eclipse: D:\eclipse-workspace\learn-hybris-tube
* Hybris\_suite: D:\training\hybris\HYBRIS\_SUITE\_1905\hybris

\* Prerequired: https://help.sap.com/viewer/3fb5dcdfe37f40edbac7098ed40442c0/1905/en-US/30a97a12eca14afbaaac88b592f5e25c.html

# Clip 1: Introduction -- DONE

* What is hybris? For eCommerce
  + Single channel
  + Multi channel
  + Cross channel
  + Omni channel
* History
  + 1997 in Switzerland then locate to German
  + Aug 1st, 2013: acquired by SAP company
* Base on Spring framework to develop. The concepts need to know:
  + Dependency Injection (Inversion of control)
  + AOP (Aspect oriented programming)
  + Spring Security
  + Spring MVC

==========

# Clip 2: Hybris b2c accelerator installation -- DONE

1. unzip the suite file and install b2c accelorator. Run cmd:

<hybris\_root>\installer> installer.bat -r b2c\_acc\_plus -A initAdminPassword=admin

2. Setting ant command. run cmd:

<hybris\_root>\hybris\bin\platform> setantenv.bat

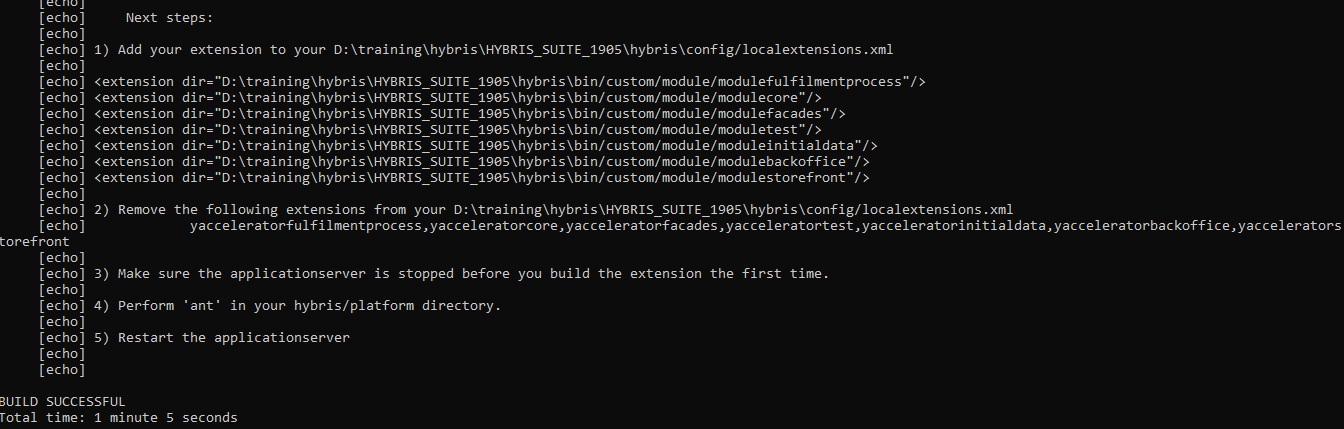
<hybris\_root>\hybris\bin\platform> ant -version // checking ant is set in env var.

se.cms.dragAndDrop

3. Generating the module for training purpose Run cmd:

<hybris\_root>\hybris\bin\platform> ant modulegen -Dinput.module=accelerator -Dinput.name=training -Dinput.package=de.hybris.training -Dinput.plate=develop

4. Following these steps listed by command after running finished.



5. Add domain name for path:

* Open the local.properties file under <hybris\_root>\hybris\config and add these line

*website.electronics.http=http://electronics.local:9001/modulestorefront*

*website.electronics.https=https://electronics.local:9002/modulestorefront*

*website.apparel-uk.http=http://apparel-uk.local:9001/modulestorefront*

*website.apparel-uk.https=https://apparel-uk.local:9002/modulestorefront*

*website.apparel-de.http=http://apparel-de.local:9001/modulestorefront*

*website.apparel-de.https=https://apparel-de.local:9002/modulestorefront*

* Open the host file under C:\Windows\System32\drivers\etc then modify:

*127.0.0.1 localhost electronics.local apparel-uk.local apparel-de.local*

6. Initialize system: database, structure, setup system. Take ~30 minutes

<hybris\_root>\hybris\bin\platform> ant initialize

7. Start site. Run cmd:

<hybris\_root>\hybris\bin\platform> hybrisserver.bat

8. Stop site. Ctrl + C on Command Line Console

==========

# Clip 3: Review the UI on init site -- DONE

1. After the site started, open browser at one of links below:

*http://electronics.local:9001/modulestorefront*

*https://electronics.local:9002/modulestorefront*

*http://apparel-uk.local:9001/modulestorefront*

*https://apparel-uk.local:9002/modulestorefront*

*http://apparel-de.local:9001/modulestorefront*

*https://apparel-de.local:9002/modulestorefront*

1. Choose one product base on category -> add to cart -> checkout -> register -> payment method -> complete order.
2. Go inside the Backoffice Administrator Cockpits to check the order list.
   1. Login at: <domain\_local>/backoffice
   2. Username: admin. Using the password which created when you installed recipe b2c\_acc\_plus as environment variable.
   3. Search “order” in the left site.
   4. Check the order which recent created in step 2.

# Clip 4: Hybris Impex Tutorial (import and export data)

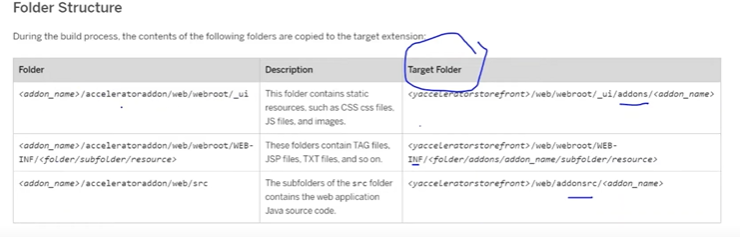
# Clip 5: Create Extension in Hybris SAP

* Open cmd at <Hybris\_root>\hybris\bin\platform and run: ant extgen
* Choose template, Input the name, input package name,
* Add new generated extension into localextensions.xml file
* Run ant clean all
* Run hybrisserver.bat

NOTE: We can change location extension generated from “custom” folder to other by opening the project.properties in \hybris\bin\platform\extgen path.

# Clip 6: Create Add On In Hybris SAP

Addon is a extension but created to solved specific function, and addon is dependence that mean it needs to integrated with other extention.



Steps create addon:

* Create new addon by creating new extension choose yaddon template
* Update localextensions.xml file
* Install add-on Run: ant addoninstall –Daddonnames=”<source\_addon\_extension\_name>” –DaddonStorefront.storefrontTemplateName=”<target\_extension\_name>”

OR:

ant addoninstall -Daddonnames=”<source\_addon\_extension\_name>” -DaddonStorefront.yacceleratorstorefront=”<target\_extension\_name>”

Example: ant addoninstall –Daddonnames=”testaddon” –DaddonStorefront.storefrontTemplateName=”hybrislearningstorefront”

* Uninstall add-on Run: ant addonuninstall –Daddonnames=”<source\_addon\_extension\_name>” –DaddonStorefront.StorefrontTemplateName=”<target\_extension\_name>”
* Run cmd under /platform folder: **ant clean all** to apply the source addon to target extension.
* Verify the source folders are created in target extension.

# Clip 7: Items.xml file

This file help us define the object model of database. In every extension, we also have <name\_ext>core-items.xml file.

## Structure Hierarchy In Order.

<items>

<atomictypes>

// This type define the core object of java system. Like Object, Number, Integer, String, Boolean, Float…

</ atomictypes>

<collectiontypes>

// Define the collection type like: List, Set, …

</ collectiontypes>

< enumtypes>

// Define enum

</ enumtypes>

<maptypes>

// Define the map with key and value

</maptypes>

<relations>

// Define the relationship between 2 item types (2 table): one to one, …

</relations>

<itemtypes>

// Define the item type will be map to the table in database.

</itemtypes>

</items>

## Some Specific notes

* autocreate=*"true"* generate=*"false"*
* This mean just created the system data object during initialization, and not create the Jalo class.

Collection Type

<maptype code=*"ExampleMap"*

argumenttype=*"Language"*

returntype=*"java.math.BigInteger"*

autocreate=*"true"*

* generate=*"false"*/>
* Argumenttype is the key, and returntype is the value for the map exampleMap code.
* <deployment table=*"UserRights"* typecode=*"29"*/>
* Table and typecode attribute have to unique, typecode is the number with range 1 -> 32767.
* From 1 -> 10000 is the range typecode for the itemType of hybris system, the remain for the custom type.
* <persistence type=*"property"*/>
* The attribute will be store into db.
* <persistence type=*"dynamic"*/>
* The attribute data will not store into db, only store in memory of server(RAM)
* <persistence type=*"property"* attributeHandler=*"beanId"*/>
* With the attributeHandler we can define a spring bean .xml file to handle value for this attribute. In this handler class, we implement the class DynamicAttributeHandler<Target, Source> /AttributeHandler<Target, Source>, to return target value.
* <modifiers read=*"true"* write=*"true"* search=*"true"* optional=*"true"*/>
* Getter or setter will be generated if read and write are true value.
* Search is true mean allow to search by flexible search
* Optional is true mean this field is not required.

# Clip 8: Flexible Search Query

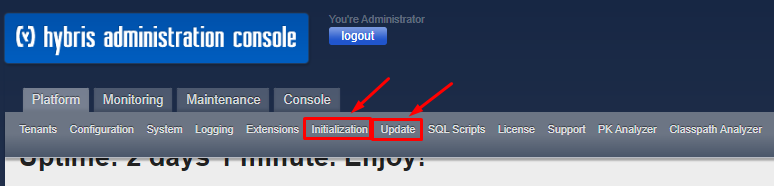
Flexible Search:

It is a Hybris build in query language base on SQL syntax.

It enables us to search the records from DB base using item types.

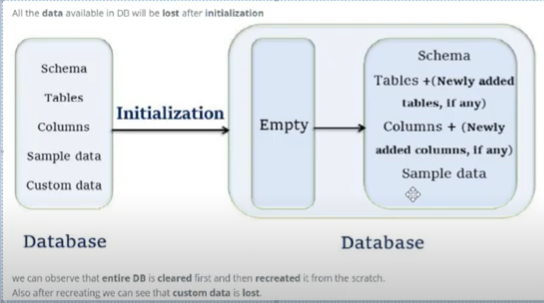
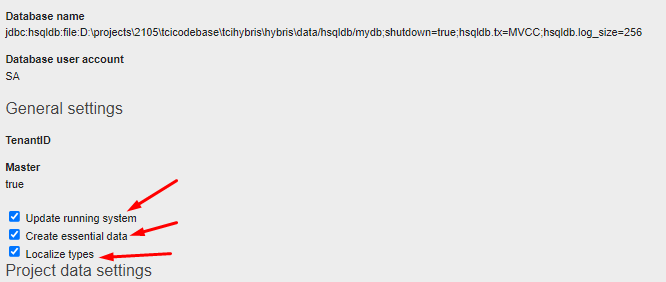
In flexible search, We never use data table name. We only use the code of the item type corresponding table.

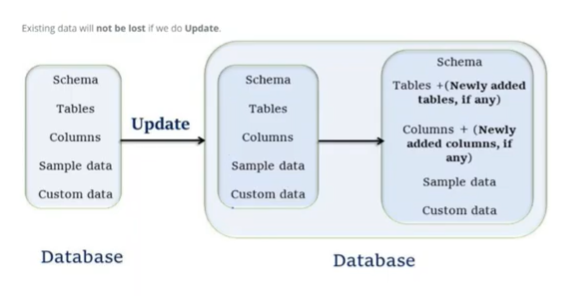
# Clip 10: system initialization and update in hybris

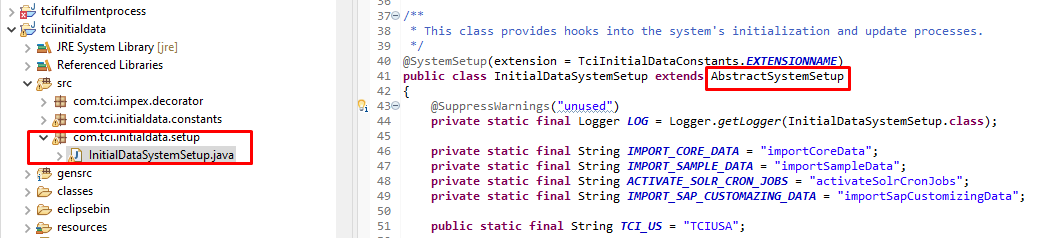


* Initialize: there are 2 approaches to do that:
  + 1: You can do it in HAC: platform -> initialization
  + 2: You can do it by running cmd in platform folder: ant initialize
* Update: There are 2 approaches to do that:
  + 1: In HAC: platform -> update then choose option base on your target updation (update running system, create essential data, localize types)
  + 2: Run cmd in platform folder: ant updatesystem –Dtenant=<tenantName> -DtypeSystemOnly=true

NOTE: 2 parameters behind the cmd in update system are optional







Using system setup to import impex data while initialize or update system.

# Clip 11: ant all and ant clean all command.

The different is ant clean all will delete all file and re-create files. Ant all is only update or create new item type or apply on the code change.

Ant all ~ ant

Ant clean all ~ ant clean

# Clip 12: Hot folder configuration in hybris -- DONE

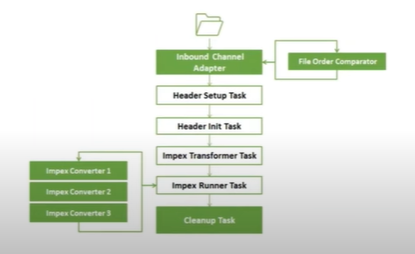
Hot folder is one of the best way to import data into Hybris system

It is based on the spring integration frameworks.

Faster (import single or multi thread), automatically, convenience than import impex from HAC



Spring configuration running flow:



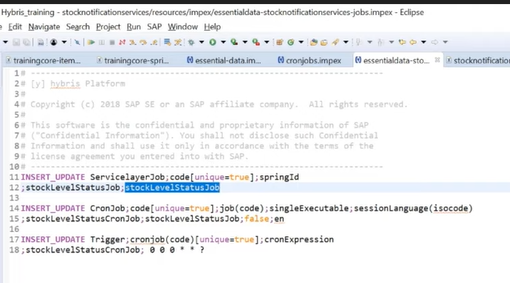
Search for \*hot-folder-\*spring.xml

Clip 13: Cronjob

To do specific task is repeated cyclically and schedule at certain interval of time.

Elements of cronjob:

* Job class
* Trigger

The job is defined to do specific task, the job combined with trigger to create an instance called a cronjob which is run at certain interval of time. 

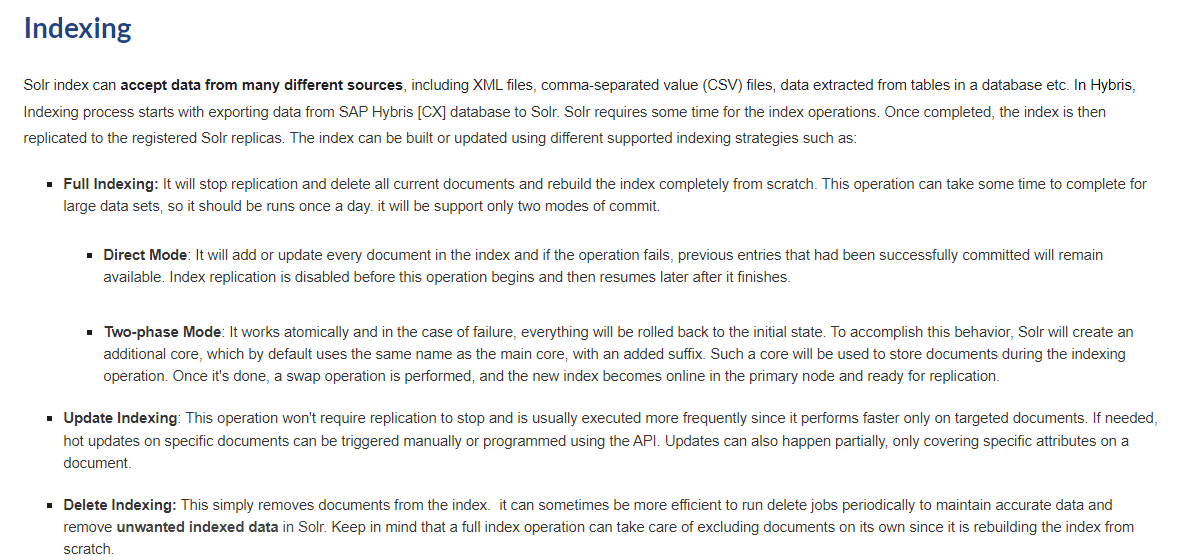
# Clip 17: Hybris Business Process

# Clip 19: Hybris Solr Search Indexing -- DONE

Overview

Solr the embed module in Hybris suite purely support index and search than consistence data to document.

Index Strategies:



Concepts in indexing:

* Index like a table in database
* Document like a row in database
* Field like a column in database

References:

<https://www.expertshybris.com/solr>

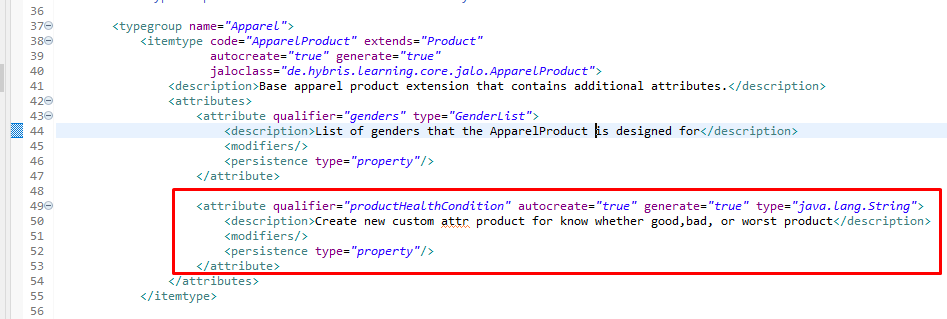
<https://help.sap.com/docs/SAP_COMMERCE/9433604f14ac4ed98908c6d4e7d8c1cc/21f689cf7c71485c9a45df03c74064e6.html?version=1905>

## Requirement:

Add new attribute into product to vote quality, and this attribute also apply to search result page. (SRP)

### Add new attribute for existed product.

Open hybrislearningcore-items.xml file



### Build and update system

### Run impex to add value for new attribute in some product base on product id.

*$productCatalog=apparelProductCatalog*

*$productCatalogName = Apparel Product Catalog*

*$catalogVersion=catalogversion(catalog(id[default=$productCatalog]),version[default='Staged'])[unique=true,default=$productCatalog:Staged]*

*$baseProduct=baseProduct(code, catalogVersion(catalog(id[default='$productCatalog']),version[default='Staged']))*

*$approved=approvalstatus(code)[default='approved']*

*$taxGroup=Europe1PriceFactory\_PTG(code)[default=eu-vat-full]*

*INSERT\_UPDATE ApparelProduct;code[unique=true];$catalogVersion;productHealthCondition*

*;300044623;;goodQuality*

*;300044624;;badQuality*

*;300045375;;worstQuality*

### Synchronize data from Stages to Online in BO

Login to BO and search product by id, then check in Staged version whether existed new attribute with new value which just imported.

* Sync manual product by product in BO
* Sync all product and content by smartedit

### Check the new attribute in Solr server

<https://localhost:8983/solr/#/master_apparel-uk_Product_default/query>

* User/password: get from the project.properites file of solrServer project.
* Search with your product id: id:apparelProductCatalog/\*/300045375
* Whether the new attribute name existed: *productHealthCondition*

### Add new attribute into solr search attribute DB.

We search for solr.impex file and update into 2 item type:

* SolrIndexedProperty
* SolrSearchQueryProperty
* Run the impex to import data for new attribute into solr

$solrIndexedType=apparel-ukProductType

INSERT\_UPDATE SolrIndexedProperty;solrIndexedType(identifier)[unique=true];name[unique=true];type(code);sortableType(code);currency[default=false];localized[default=false];multiValue[default=false];useForSpellchecking[default=false];useForAutocomplete[default=false];fieldValueProvider;valueProviderParameters[map-delimiter=|];ftsPhraseQuery[default=false];ftsPhraseQueryBoost;ftsQuery[default=false];ftsQueryBoost;ftsFuzzyQuery[default=false];ftsFuzzyQueryBoost;ftsWildcardQuery[default=false];ftsWildcardQueryType(code)[default=POSTFIX];ftsWildcardQueryBoost;ftsWildcardQueryMinTermLength

;$solrIndexedType; productHealthCondition ;text ; ; ; ; ;TRUE;TRUE; ; ;TRUE;80 ;TRUE;40 ;TRUE;20; ; ; ;

INSERT\_UPDATE SolrSearchQueryProperty ; indexedProperty(name, solrIndexedType(identifier))[unique=true] ; searchQueryTemplate(name, indexedType(identifier))[unique=true][default=DEFAULT:$solrIndexedType] ; facet ; ftsPhraseQuery[default=false] ; ftsPhraseQueryBoost ; ftsQuery[default=false] ; ftsQueryBoost ; ftsFuzzyQuery[default=false] ; ftsFuzzyQueryBoost ; ftsWildcardQuery[default=false] ; ftsWildcardQueryType(code)[default=POSTFIX] ; ftsWildcardQueryBoost ; ftsWildcardQueryMinTermLength ; includeInResponse[default=true] ; useForHighlighting[default=false]

; productHealthCondition:$solrIndexedType ; ; ; TRUE ; 80 ; TRUE ; 40 ; TRUE ; 20 ; ; ; ; ; ;

In BO, Check in facet index type whether new properties is existed.

### Add new bean to proceed facet indexing for new field.

To sync data from step 6 into solr server, we need to do the following steps below:

* Open file: commerceservices-spring-solrfacetsearch.xml
* Search to populator: *defaultCommerceSearchTextPopulator*
* Add new bean for product health:

<bean class=*"de.hybris.platform.commerceservices.search.solrfacetsearch. querybuilder.impl.DefaultFreeTextQueryBuilder"*>

<property name=*"propertyName"* value=*"productHealthCondition"* />

<property name=*"boost"* value=*"90"* />

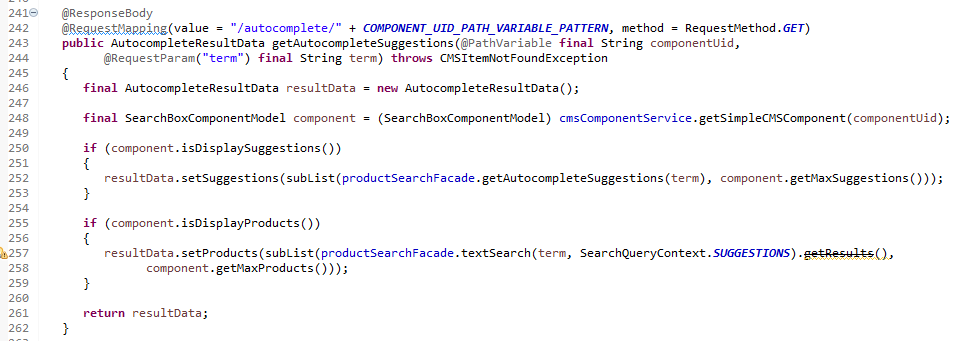
</bean>

* Run: ant all -> start server -> search facet in BO to full index to apply search new property on all products.
* NOTE: instead of go to facet to run full index, we can go to cronjob section then search for index. In the list result, we can choose the job which want to run (full, update, delete)

After facet successfully, go to solr server and search our new field in any products and products update new value, then to go storefront search with the value we imported.



SearchPageController will handle our request for search autocomple product.

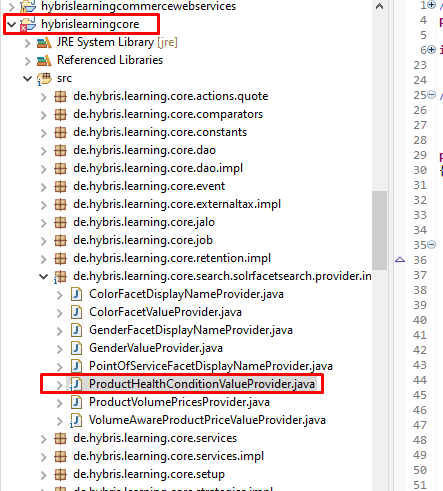


## Custom value for solr search field

In solr document, instead of use the value same with database or want create new field value base on the current data, we can custom with diff value but have the same meaning.

### Create provider for field

From OOTB, there are many providers are created under core extension. Let’s create new provider for our new field.



Process data inside:



### Register provider to bean

To get field names we need to use the *solrFieldNameProvider* , So we need to define this property in registration bean as well.

* Open the hybrislearningcore-spring.xml file and put the bean below:

<bean id=*"productHealthConditionValueProvider"* class=*"de.hybris.learning.core.search.solrfacetsearch.provider.impl.ProductHealthConditionValueProvider"* parent=*"abstractPropertyFieldValueProvider"*>

<property name=*"fieldNameProvider"* ref=*"solrFieldNameProvider"*/>

</bean>

* Build and run server: ant -> hybrisserver.bat debug

### Update impex into SolrIndexProperty type

After server start up, go into the hac and open the impex import console and run the snippet code below:

*$solrIndexedType=apparel-ukProductType*

*UPDATE SolrIndexedProperty; solrIndexedType(identifier)[unique=true]; name[unique=true]; fieldValueProvider*

*;$solrIndexedType; productHealthCondition ; productHealthConditionValueProvider*

### Run facet full index

Go inside the BO and run full index facet under apparel-uk solr.

We can debug to know more detail in our custom provider.

### Check the result

* Open the solr server to verity
* Open the storefront to verity with the replace search text (good product, bad product, worst product)

### NOTEs

In the provider we have the IndexConfig parameter in function getFieldValues, This parameter is gotten target data which populated data from SolrFacetSearchConfig model.

## Add Synonym/Stopwords/Keyword\_redirect text

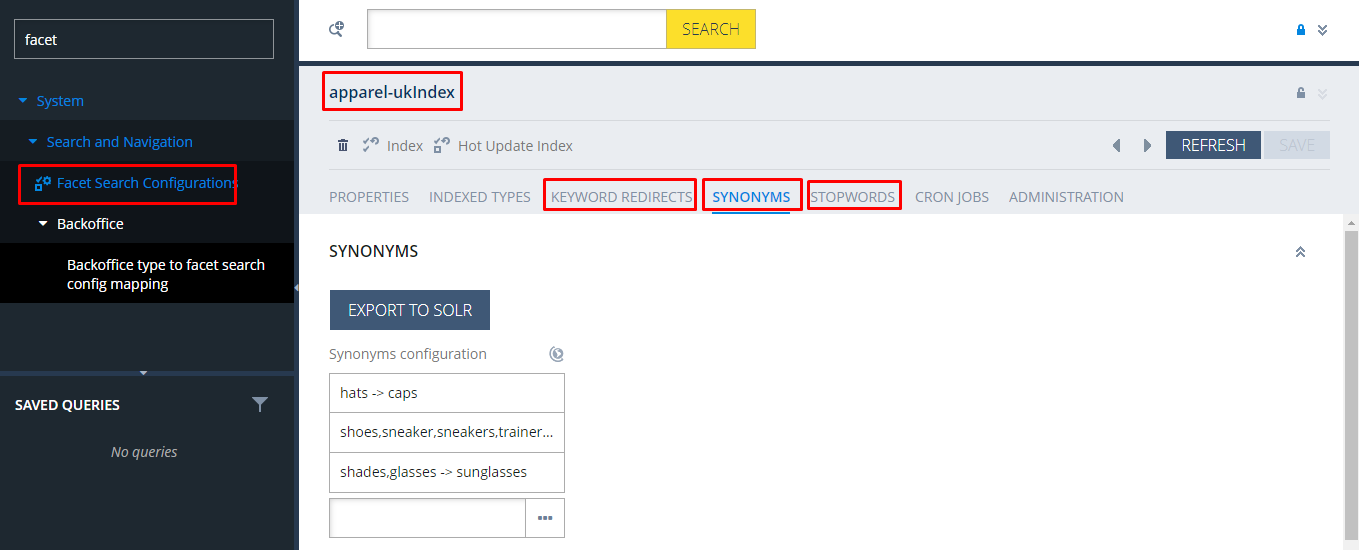
In the reality, we have to allow customer search product by many texts which have synonym meaning. So how to do that in solr search, let’s go.

### Meaning:

Go into the storefront and type “cat”, there are multiple result display. Why and how to do that.

* Synonym: with different text but have the same meaning will display same result
* Stopwords: when match the text, the search function(auocomplete) will stop
* Keyword redirect: when match the text, the redirect page with work after pressing search.

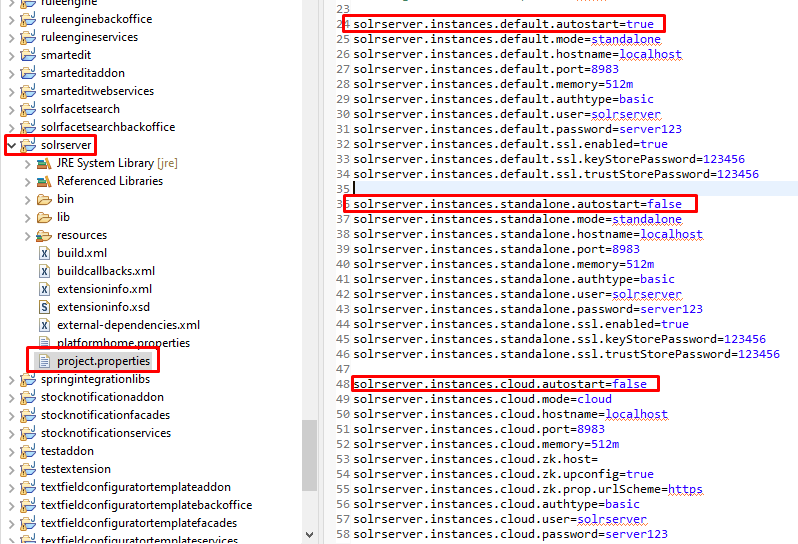
### Place to check



### Configuration

We can config in BO, but to keep consistent data between environments (dev, stg, prd) and to import with the large data, we need to store in impex data file.

## Solr server start/stop



### Start/Stop standalone

Start at: D:\projects\2105\tcicodebase\tcihybris\hybris\bin\modules\search-and-navigation\solrserver\resources\solr\8.11\server\bin

$ solr.cmd start -p 8983

$ solr.cmd stop -p 8983

### Remove node solr by cmd

bin/solr.cmd delete [-c name] [-deleteConfig true|false] [-p port]

### Solr Schema generation:

COPY D:\projects\2105\tcicodebase\tcihybris\hybris\config\solr\instances\default\configsets\default\conf\schema.xml

D:\projects\2105\tcicodebase\tcihybris\hybris\bin\modules\search-and-navigation\solrserver\resources\solr\8.11\server\server\solr\configsets\default\conf

### Synonyms config file

COPY D:\projects\2105\tcicodebase\tcihybris\hybris\config\solr\instances\default\configsets\default\conf\synonyms.txt

D:\projects\2105\tcicodebase\tcihybris\hybris\bin\modules\search-and-navigation\solrserver\resources\solr\8.11\server\server\solr\configsets\default\conf

COPY D:\projects\2105\tcicodebase\tcihybris\hybris\config\solr\instances\default\configsets\default\conf\lang\synonyms\_en.txt

D:\projects\2105\tcicodebase\tcihybris\hybris\bin\modules\search-and-navigation\solrserver\resources\solr\8.11\server\server\solr\configsets\default\conf\lang

COPY D:\projects\2105\tcicodebase\tcihybris\hybris\config\solr\instances\default\configsets\default\conf\lang\synonyms\_ja.txt

D:\projects\2105\tcicodebase\tcihybris\hybris\bin\modules\search-and-navigation\solrserver\resources\solr\8.11\server\server\solr\configsets\default\conf\lang

# Clip 20: **hybris promotion engine tutorial**

# Clip 23: Hybris OCC | OCC Swagger -- DONE

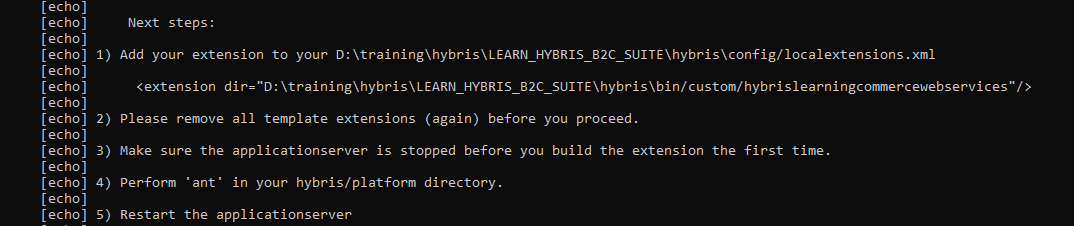
Hybris OCC is Omni Channel Connect, is used to expose functionality as stateless API. It makes power to platform and make it omni channel, and with this, we can expose and connect Hybris API to External Application like native app, native web, etc with minimum customizations.

Rest webservice.

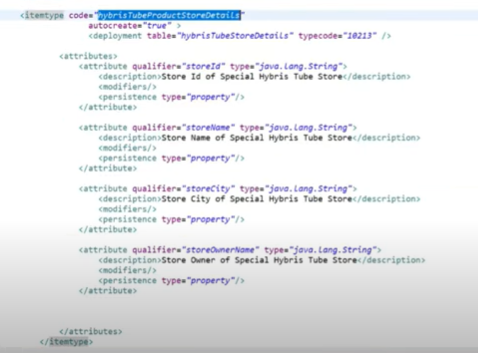
Steps to create OCC Webservice

## Create extension which used ycommercewebservices template.

* Create extension: ant extgen –Dinput.template=”ycommercewebservices” –Dinput.name=” hybrislearningcommercewebservices” –Dinput.package=”de.hybris.learning”
* Remove OOTB webservice extension: ycommercewebservices and ycommercewebservicestest
* Add entry for new extension in localextension.xml
* Import hybrislearningcommercewebservices ext into eclipse



## Create HybrisProductStoreDetail item type in hybrislearningcommercewebservices ext.



* Do ant all to generate the model
* Check the model is generated
* Create the impex to insert sample data in projectdata-\*.impex file

***INSERT\_UPDATE*** *HybrisProductStoreDetails ; storeId[unique=true] ; storeName ; storeCity; storeOwnerName*

*; S001; Uniqlo ; HCMC ; Akira Toriyama*

*; S002; Vinfast ; Ha Noi ; Pham Nhat Vuong*

*; S003; Meta ; Silicon Valley ; Mark Jackurbuck*

*; S004; MP Food ; Vinh Long Province ; Nguyen Minh Hai*

* Start the server and check data in new item type.

## Create DAO to get data for store product

* Create Dao package in hybrislearningcore ext
* Create interface HybrisStoreDAO which extends Dao from serviceLayer Platform, then add function getHybrisStoreDetailById
* Create new package and class to implement is HybrisStoreDAOImpl
* Process function getHybrisStoreDetailById
* Add HybrisStoreDAOImpl class into spring.xml file

## Create service layer

* Create package, interface and class in core ext
* Using resource from hybrisStoreDAO bean to get data.
* @Resource
* **private** HybrisStoreDAO hybrisStoreDAO;
* Register service as bean in spring.xml file

## Create DTO object in façade ext.

* Open the \*facade-bean.xml file and create object HybrisStoreData with 3 field (store city, store name, and store owner name)
* Do ant all
* Check the data model whether created.

## Create populator

* In façade ext. Create HybrisStorePopulator implement populate with target is HybrisStoreData and source is HybrisProductStoreDetailsModel

**public** **class** HybrisStorePopulator **implements** Populator<HybrisProductStoreDetailsModel, HybrisStoreData>

* Populate data for the target object.

## Create converter

* In façade ext, open \*facades-spring.xml file and add following configurations to create converter;

<bean id=*"hybrisStoreConverter"* parent=*"abstractPopulatingConverter"*>

<property name=*"targetClass"* value=*"de.hybris.learning.facades.store.data.HybrisStoreData"*/>

<property name=*"populators"*>

<list>

<ref bean=*"hybrisStorePopulator"*/>

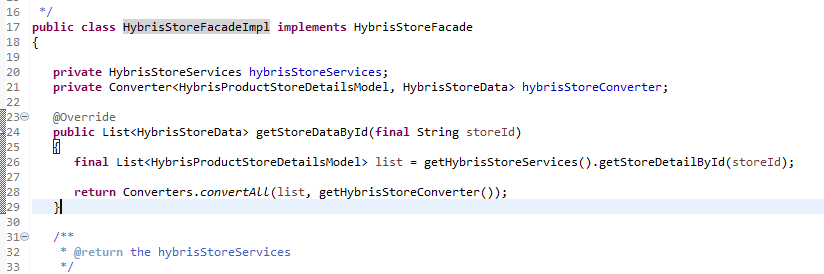
</list>

</property>

</bean>

<bean id=*"hybrisStorePopulator"* class=*"de.hybris.learning.facades.populators.HybrisStorePopulator"* />

## Create Façade layer

* In façade ext, create façade package: de.hybris.learning.facades.hybrisStore
* Create the interface: HybrisStoreFacade.java , with one prototype: List<HybrisStoreData> getStoreDataById(String storeId);
* Create the new package and class implement: HybrisStoreFacadeImpl.java
* 
* Register façade class into \*façade-spring.xml file

<bean id=*"hybrisStoreFacade"* class=*"de.hybris.learning.facades.hybrisStore.impl.HybrisStoreFacadeImpl"* >

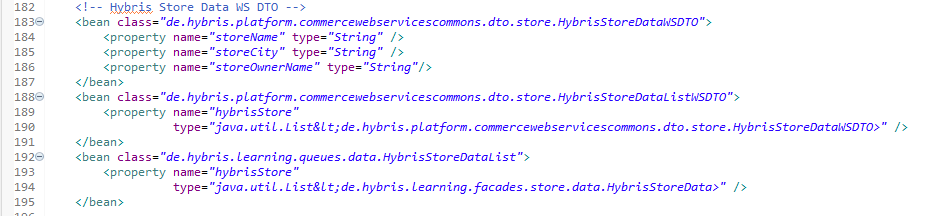
<property name=*"hybrisStoreServices"* ref=*"hybrisStoreServices"*/>

<property name=*"hybrisStoreConverter"* ref=*"hybrisStoreConverter"*/>

</bean>

## Create Controller layer In \*commercewebservices ext

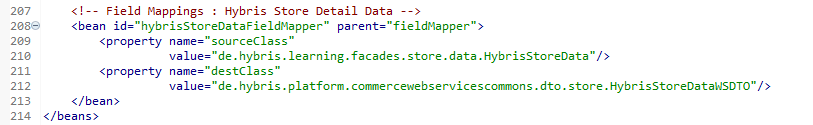
* Create some DTO in hybrislearningcommercewebservices-beans.xml to mapping data from façade layer.

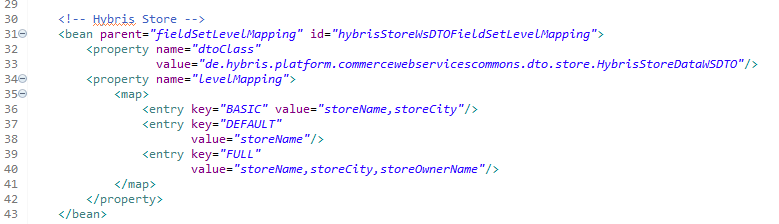


* Do ant to generate the DTO model in platform folder.
* Under web/src folder, create new file name: HybrisStoreController.java in de.hybris.learning.v2.controller package.



* In the return data, we have step mapping data from list of HybrisStoreData objects to list of HybrisStoreDataWSDTO. To do that, we need define a mapping in dto-mapping\*.xml file



* We have 3 values response type: Basic, Default, and Full. In each type, we will define the response diff value. In the dto-level-mapping-\*.xml add:
* 

## Adding user to authentication on API

* There are 3 main role to authenticate.
  + ROLE\_TRUSTED\_CLIENT
  + ROLE\_CUSTOMERGROUP
  + ROLE\_CLIENT
* Insert data for user access the API, update the projectdata-hybrislearningcommercewebservices.impex file.

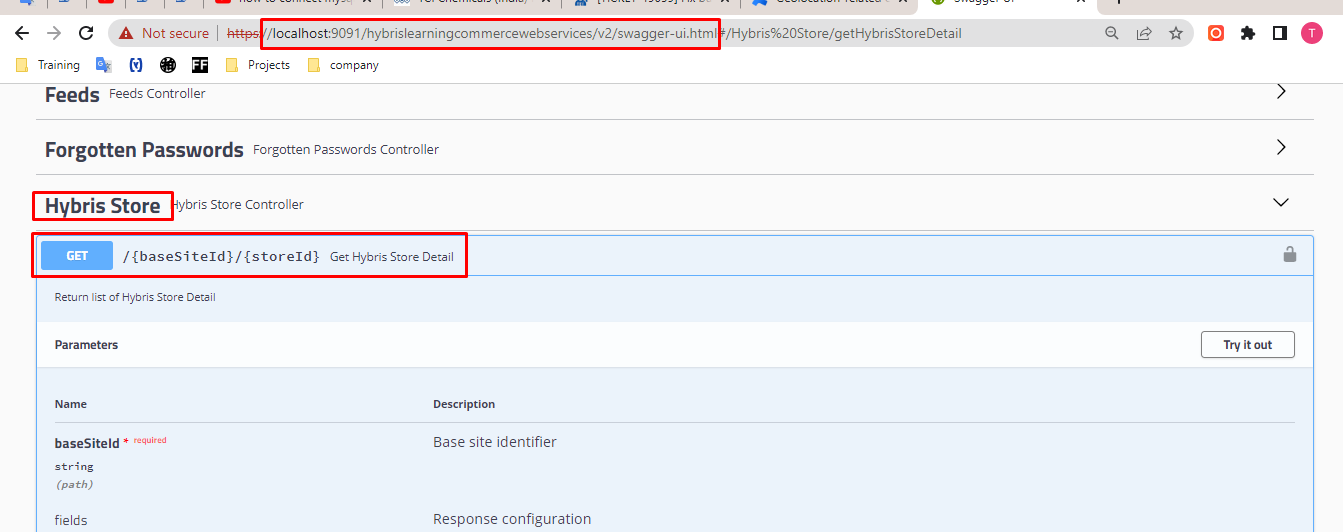
**INSERT\_UPDATE** OAuthClientDetails ; clientId[unique=true] ; resourceIds ; scope ; authorizedGrantTypes; clientSecret ; authorities

; hybrisTube; hybris; basic ;authorization\_code, refresh\_token, password, client\_credentials ; hybrisTube; ROLE\_TRUSTED\_CLIENT

OAuthClientDetails;clientId[unique=true];resourceIds;scope;authorizedGrantTypes;authorities;clientSecret;registeredRedirectUri ;myclientId;hybris;basic;authorization\_code,refresh\_token,password,client\_credentials;ROLE\_CLIENT;myclientpassword;<http://localhost:9001/authorizationserver/oauth2_callback;>

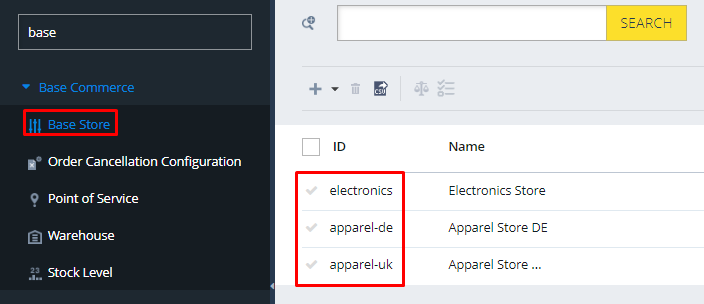
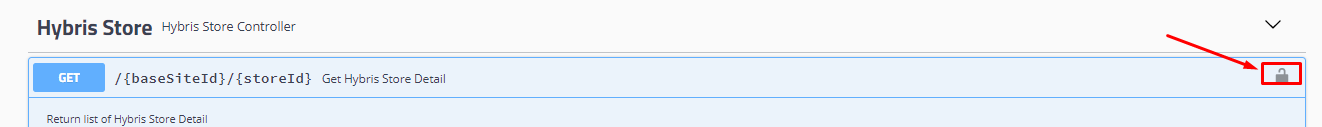
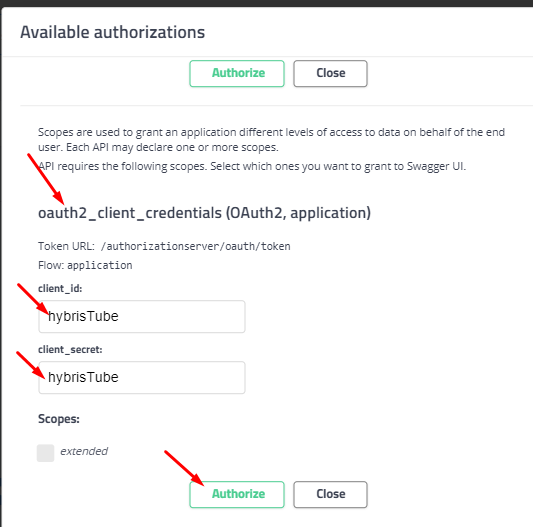
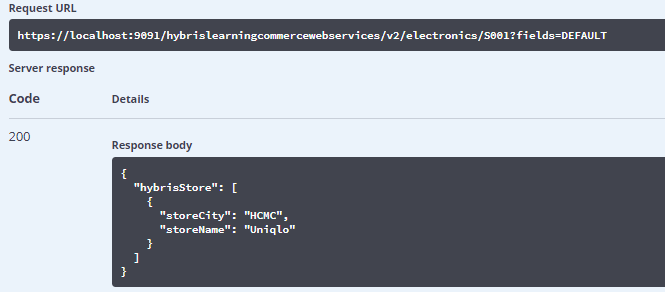
## Ant all and start server to verify.

* Do ant all to build project
* Do hybrisserver.bat debug to start server in debug mode.
* After the server started. Check the api is existed in swagger ui. Domain\_name + port + nameext+ v2 + swagger-ui.html
  + Eg: <https://localhost:9091/hybrislearningcommercewebservices/v2/swagger-ui.html>



* Verify data added in impex file from step 2, and step 10 in HAC. If data is not exised, the system required update on \*webservices ext to import data from projectdata.

## Test our API

* We can use post main or test directly on swagger browser cause our api is using method GET.
* 1: Input the parameters
  + BaseSiteId: in BO we have 3: electronics, apparel-de, apparel-uk
  + 
  + Fields: we choose: Default
  + storeId: we choose: S001 (data imported from step 2)
* 2: execute
  + We got the access denied issue cause of we need have right permission.
* 3: Get token for authenticating
  + Click on lock icon
  + 
  + Input client id and client secret (data imported from step 10)
  + 
* 4: execute again. The response is correct.
* 

Congratulation!. Our first API has done.

# Clip 28: Hybris Email Flow -- DONE

In this clip we will create an example about email a discount 20% voucher code to user when they purchase products with total price is more than $500.

## Create a HybrisEmailProcess Item Type.

<itemtype code=*"HybrisEmailProcess"* autocreate=*"true"* generate=*"true"* extends = *"StoreFrontCustomerProcess"*

jaloclass=*"de.hybris.learning.core.jalo.HybrisEmailProcess"*>

<attributes>

<attribute type=*"AbstractOrder"* qualifier=*"cart"*>

<persistence type=*"property"* />

</attribute>

</attributes>

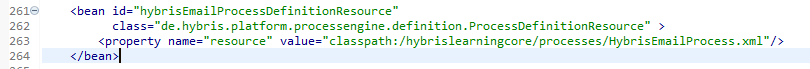
</itemtype>

* Do ant all to generate the jalo class and model of HybrisEmailProcess item type

## Give entry of “hybrisEmailProcess.xml” in “hybrislearningcore-spring.xml” file

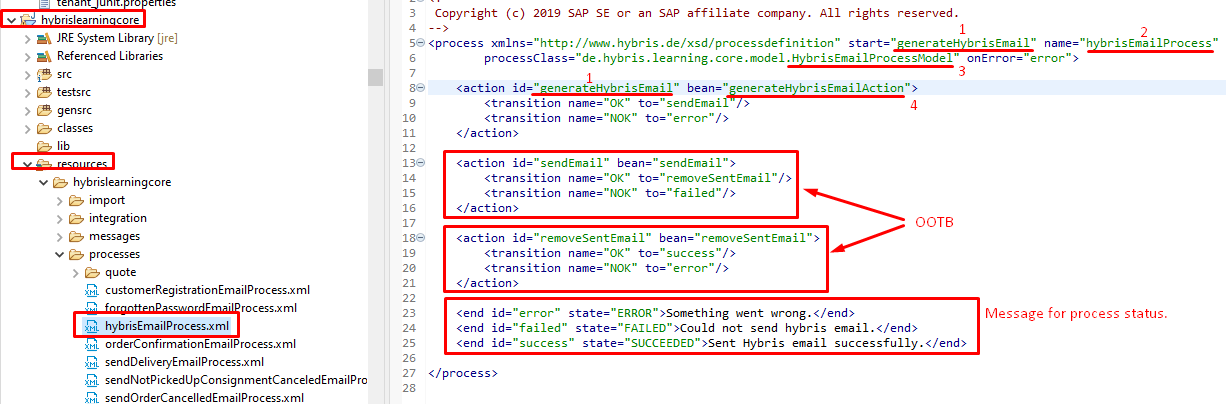
The hybrisEmailProcess.xml file store the steps process for sending email, we need to register this file into context.

* In the process resource definition section:



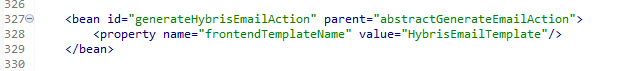
## Create a file with name HybrisEmailProcess.xml

* Under hybrislearningcore. Create hybrisEmailProcess.xml file



* 1: The name of start process
* 2: The name of our process, this name used to determine among others.
* 3: The model of our process which create in step 1
* 4: The bean proceeds our actions on this process. Let’s create it in next step

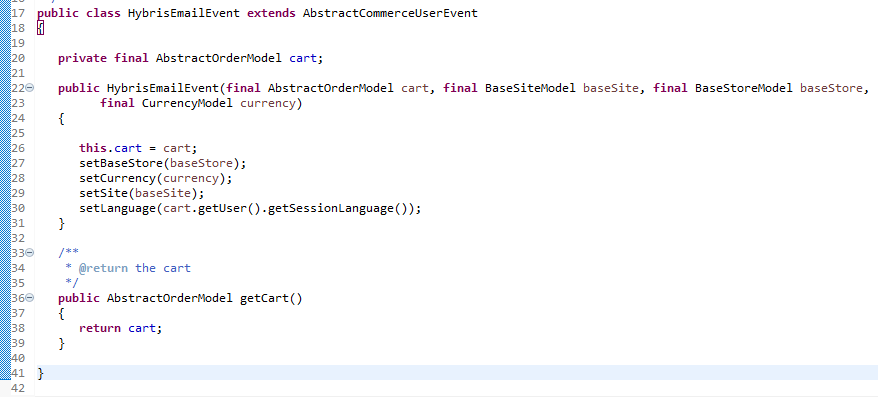
## Define and register the generateHybrisEmailAction bean in hybrislearningcore-spring.xml file



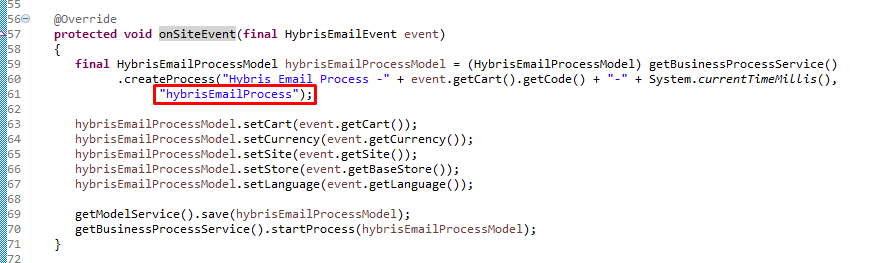
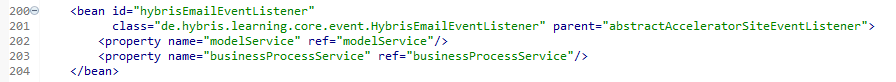
* HybrisEmailTemplate will be defined later. This template to display the content email.

## Create event class.

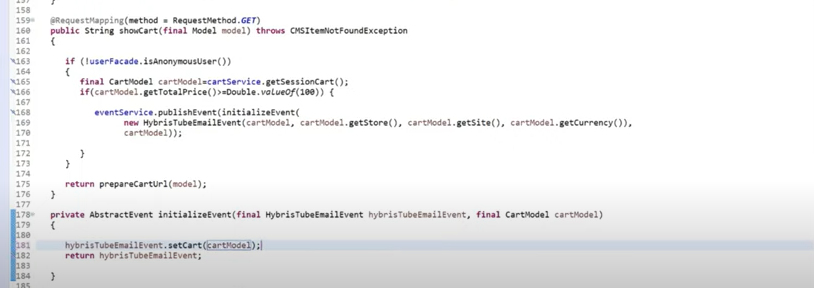
* Under src/ folder, under event package, create class name “HybrisEmailEvent.java” extends AbstractCommerceUserEvent



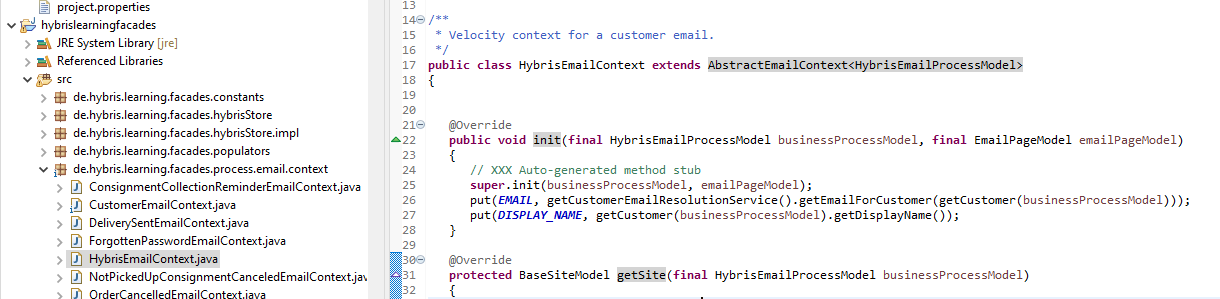
## Create event listener class and mapping into bean

* Under src/ folder, under event package, create “HybrisEmailEventListener.class” extends AbstractAcceleratorSiteEventListener
* Proceed onSiteEvent function
* 
* NOTE: the second argument in the createProcess function is the name of process.
* Register the listener class into \*spring-bean.xml
* 

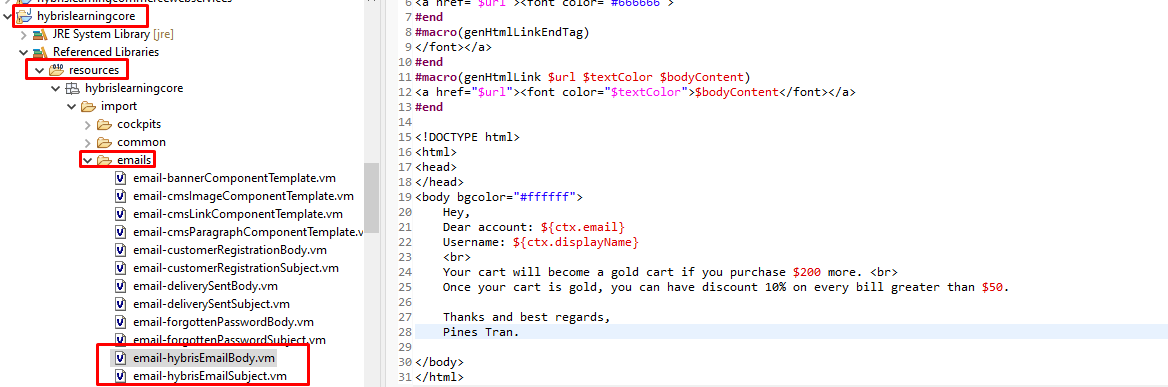
## Change code in CartPageController.java to call event HybrisEmailEvent.java

* Find to function showCart and update check logic then publish event.
* 

## Create HybrisEmailContext.java file

* This file is used to transfer data between process model to vm file template. Required implement AbstractEmailContext
* In façade ext, find to \*hybrislearning/\*context.java, copy the CustomEmailContext.java file -> paste in the same package, rename to HybrisEmailContext.java
* Update the Generic model to HybrisEmailProcessModel
* Remove all content insite then generate the methods need to implement
* Generate override init method from AbstractEmailContext
* Transfer 2 data: email and display name
* 
* Register HybrisEmailContext.java into \*facade-spring.xml file
* 

## Create vm file for Subject and Body email

* Body content
* 
* Subject content
* Congratulation!!! ${ctx.displayName}
* The “DisplayName” and “email” are the data transferred by “HybrisEmailContext.java” file

## Create the impex data to mapping with the email template from step 4

Please refer to: hybris-email-data.impex file

## Deploy and update system

* Do ant all
* Do hybrisserver.bat debug
* Go to HAC update for the new item type applied to DB

## Test our function

* Go to storefront add to cart items have total price more than 200$ then checkout then check mail

## NOTEs: Multi language for email

# Clip 33: Hybris Backoffice Customization

## Create new extension for backoffice.

* Open cmd at /hybris/bin/platform
* Run $ ant extgen
  + Type: ybackoffice
  + Type your ext name: hybrisTubeBackofficeExtention
  + Type your package name: de.hybris.training
  + Others press enter to generate default like: related exts, sample data, …
* After generate successfully, add the new name extension into localextension.xml file under config folder.

## Import ext into eclipse.

Copy the path link to the new custom extension: D:\training\hybris\HYBRIS\_SUITE\_1905\hybris\bin\custom\hybrisTubeBackofficeExtension

Open eclipse -> import project -> paste the path above at source -> done

## Explore the backoffice config file.

1. Add entry of new tab key, New attribute Section key.

# Clip 52: Where value of Localized attributes is Saved in Hybris DB





# Clip 60: Run Multiple Hybris Instance in One Machine

We need to add more configuration into local.properties file under /hybris/config folder:

* Update the debug port:

tomcat.debugjavaoptions=-Xdebug –Xnoagent –Xrunjdwp:transport=dt\_socket,server=y,address=8001,suspend=n –Xverify:none –DforceANSI=true

* Update port tcp/ip:

tomcat.http.port=9090

tomcat.ssl.port=9091

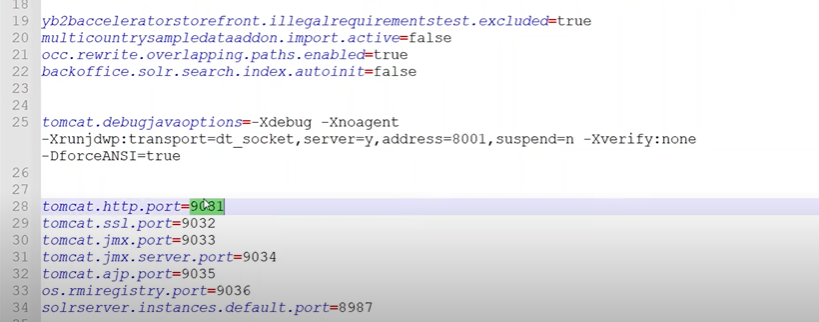
tomcat.jmx.port=9092

tomcat.jmx.server.port=9093

tomcat.ajp.port=9035

os.rmiregistry.port=9036

solrserver.instances.default.port=8987

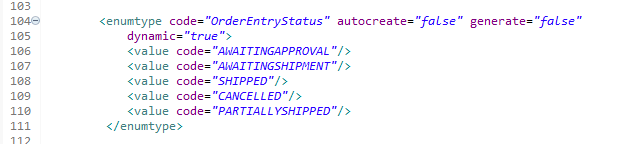


* Run cmd to apply modify to server tomcat: ant clean all
* Start server: hybrisserver.bat

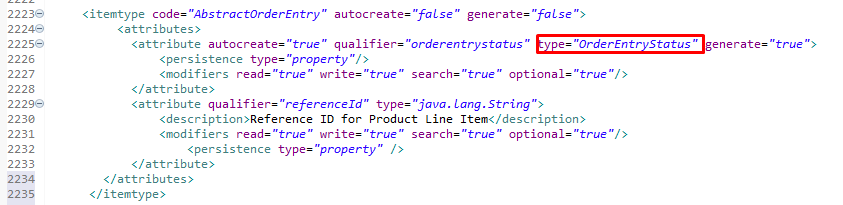
# Clip ex1: Localization for enum type

As usually we can localized for attribute in item type, we can also localize on enum type.

## Create a OrderEntryStatus enum type

* This enum is used to define status of order
* In the \*core-items.xml file add these code below:
* 
* To know more about structure of \*-items.xml file, please refer to [Clip 7](#_Clip_7:_Items.xml)

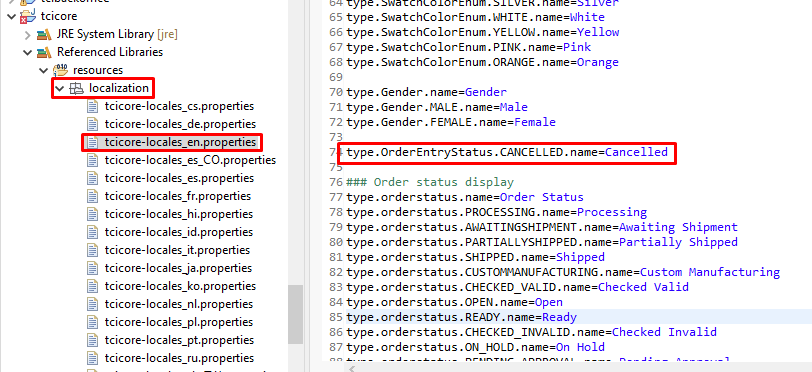
## Create a item type contain the enum in step 1



## Process to get localized data.

* We use the “EnumerationService” service from OOTB
* Inject service:
* @Resource(name = "enumerationService")
* **private** EnumerationService enumerationService;
* Implement method:
* enumerationService.getEnumerationName(OrderEntryStatus.CANCELLED, Locale.ENGLISH);
  + The second argument can leave empty to get the default locale from site.

## Add localized data in Localized folder

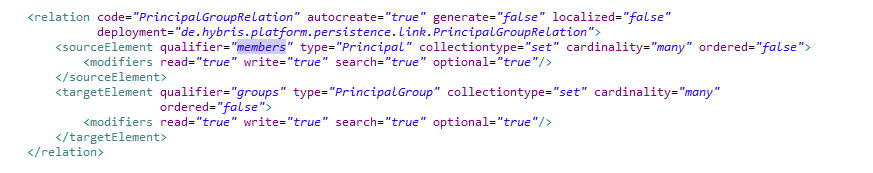
* Add all enum data for language you desired base on the \*.properties file
* 

## Verify

* Once you run ant
* Go into HAC to update localized
* Go to the BO -> find to Types -> search OrderEntryStatus -> in here we can find out all localized data we input for each enum item
* On the storefront we can debug or we can create the UI to display.

# Clip ex2: flexible search Join Tables

We need to join tables in the hyerachy



*SELECT {c:UID}*

*FROM {*

*B2BCustomer AS c*

*JOIN PrincipalGroupRelation AS rel ON {rel:SOURCE}={c:PK}*

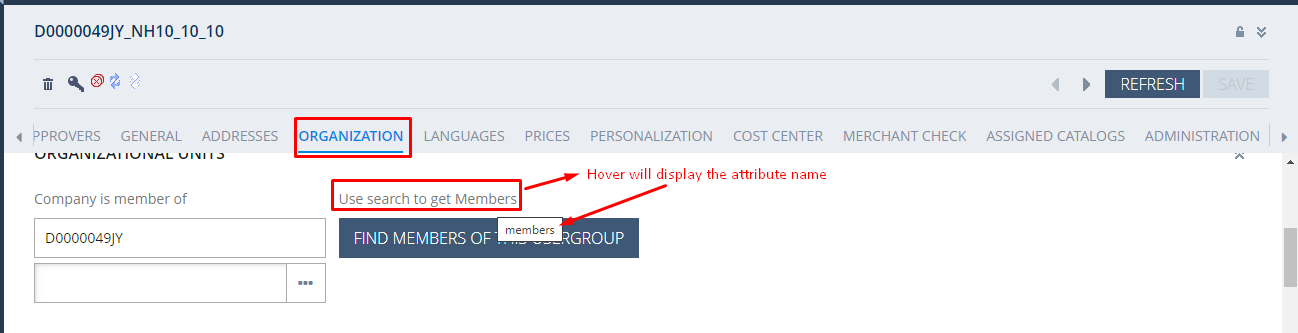
*JOIN B2Bunit AS u ON {u:pk}={rel:target}*

*}*

*WHERE {u:UID} like '%\_NH00%'*

**Explain**:

Tips in BO:



# Clip ex3: Get environment variable

To make dynamic data, we often set env variables in \*.properties file

Today we will discover 3 ways to get env var.

## From .xml file

In \*-spring.xml file, in place where define the bean, add property like below:

<property name="domainWhitelist" value="#{configurationService.configuration.getProperty('domain.external.whitelist')}" />

In bean java, define domainWhitelist variable with setter and getter

## From configuration service

Implementation:

// inject:

**import** de.hybris.platform.servicelayer.config.ConfigurationService;

@Resource(name = "configurationService")

**private** ConfigurationService configurationService;

// call:

configurationService.getConfiguration().getString(***DEFAULT\_COUNTRY\_ISO\_PROP***, ***DEFAULT\_COUNTRY\_ISO***);

## From siteConfigService

This config refer to get property for basesite id.

Implementation:

// inject:

import de.hybris.platform.acceleratorservices.config.SiteConfigService;

@Resource(name = "siteConfigService")

private SiteConfigService siteConfigService;

// Call:

siteConfigService.getProperty(LyonscgseoaddonWebConstants.DOMAIN\_NOINDEX);

The method getProperty(java.lang.String) is implemented below:

@Override

**public** String getProperty(**final** String property)

{

**final** BaseSiteModel currentBaseSite = getBaseSiteService().getCurrentBaseSite();

Assert.*notNull*(currentBaseSite, "BaseSite should not be null");

**final** Configuration configuration = getConfigurationService().getConfiguration();

**final** String currentBaseSiteUid = "." + currentBaseSite.getUid();

**final** UiExperienceLevel uiExperienceLevel = getUiExperienceService().getUiExperienceLevel();

String uiExpLevel = uiExperienceLevel != **null** ? "." + uiExperienceLevel.getCode() : StringUtils.***EMPTY***;

**return** configuration.getString(property + currentBaseSiteUid + uiExpLevel,

configuration.getString(property + currentBaseSiteUid,

configuration.getString(property + uiExpLevel, configuration.getString(property, **null**))));

}

## From Config static

// Import

import de.hybris.platform.util.Config;

// call:

Config.getParameter(ALL\_COUNTRY\_CODES);

## Get directly in JSP

<c:set var="yourConfig" value="${jalosession.tenant.config.getParameter('your.key.config')}" />

## Get in JS global fields

# Clip ex4: OOTB Services

## Get current currency

**import** de.hybris.platform.servicelayer.i18n.CommonI18NService;

**import** de.hybris.platform.core.model.c2l.CurrencyModel;

**final** CurrencyModel currentCurrencyModel = commonI18NService.getCurrentCurrency();

CurrencyModel currencyModel = getSessionService().getAttribute("currency");

## Get current user

**import** de.hybris.platform.servicelayer.user.UserService;

**import** de.hybris.platform.core.model.user.UserModel;

**final** UserModel currentUserModel = userService.getCurrentUser();

## Get current country

**import** de.hybris.platform.servicelayer.session.SessionService;

**import** de.hybris.platform.core.model.c2l.CountryModel;

CountryModel currentCountry = sessionService.getAttribute("country");

currentCountry = commonI18NService.getCountry("AU");

# Clip ex5: JOB and cron-job

Step 1: Create model extends CronJob

<itemtype code="tciAzureTokenCronJob" autocreate="true" generate="true"

jaloclass="com.tci.core.jalo.TciAzureTokenCronJob" extends="CronJob">

<attributes>

<attribute qualifier="days" type="java.lang.Integer">

<description>Token will be expired in days.</description>

<persistence type="property" />

<modifiers write="true" read="true" search="true" optional="false"/>

</attribute>

</attributes>

</itemtype>

Step 2: Create Job Class extend AbstractJobPerformable<TciAzureTokenCronJobModel>

Step 3: Regiser Job class into spring context.

<bean id="tciAzureBlobTokenJob" class="com.tci.azureblob.job.TciAzureBlobTokenJob" parent="abstractJobPerformable"/>

Step 4: Create impex to Map Job spring id into ServiceLayer

INSERT\_UPDATE ServicelayerJob;code[unique=true];springId[unique=true]  
;tciAzureBlobTokenJob;tciAzureBlobTokenJob

Step 5: Create impex to create Cronjob and Trigger (if any)

# TCI Azure Blob Token Job

INSERT\_UPDATE TciAzureTokenCronJob; code[unique=true];job(code);days;singleExecutable;sessionLanguage(isocode)

;tciAzureTokenCronJob;tciAzureBlobTokenJob;3;false;en

INSERT\_UPDATE Trigger;cronjob(code)[unique=true];cronExpression;active[default=false]

; tciAzureTokenCronJob;0 30 7 1/1 \* ? \* ;

# Clip ex6: Add maven dependencies

Step 1: get and add dependencies into external-dependencies.xml file

Step 2: Update usemaven=”true” in extension tag of extensioninfo.xml file

Step 3: Add unmanaged-dependencies.txt file and list down the .jar files under /lib folder which you don’t want maven remove when build.

Step 4: ant to build.

# Clip ex7: Custom Condition Promotion Rule

# Clip ex8: Apply multiple promotion rule